REMARKS

Claims 1-21 remain in this application and are pending for reconsideration.

Claims 1-7, 9-10, and 20-21 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Clark (U.S. Patent No. 6,598,068) in view of Oliver (U.S. Patent No. 6,029, 190). Claims 8 and 11-19 were rejected under 35 U.S.C. § 103(a) as being unpatentable over Clark in view of Oliver and further in view of Tillier (U.S. Patent No. 6,421,742). The Applicants respectfully traverse these rejections based on the following remarks.

Clark discloses a method and apparatus for managing how threads of a multi-threaded program share a resource. One thread of a program is given priority over other threads of the program by granting to the thread possession of the lock associated with the resource regardless of whether the thread currently requires use of the resource. The other threads are designed to indicate to the priority thread when they require use of the resource. If the priority thread is done using the resource and detects that another thread is waiting to use the resource, the priority thread releases the resource lock for the resource. After releasing the resource lock, the priority thread automatically requests the resource lock. After using the resource, any non-priority thread releases the resource lock to the priority thread if the priority thread has requested the resource, without regard to whether any other threads may be waiting for the resource.

The Applicants respectfully submit that the Clark patent fails to disclose or suggest multiple processors arranged to access a shared resource. Although the Examiner has asserted that expanding a typical single-processor environment into a multi-processor environment is a simple transition, the Applicants respectfully submit that this type of expansion of the Clark disclosure would not have been obvious at the time of the present invention. Further, the Applicants respectfully submit that Clark does not disclose or even suggest that an update or change of state may be made by a single update thread only when none of the multiple worker threads are processing work on the shared resource, and does not disclose or even suggest multiple worker threads that are able to perform work on the shared resource concurrently. As disclosed by Clark, when the resource lock 410 is released by the main thread 418, the operating system 412 transfers ownership of the resource lock 410 to one of the worker threads that is waiting to use the single-access resource 408. The worker thread 414 that is granted ownership of the resource lock 410 performs the steps illustrated in FIG 7 (see column 10, lines 55-61). Further, the worker thread 414 acquires the resource lock at step 700, and at step 702, the worker thread 414 uses the resource 408 to perform a task. When it is through using the resource 408, the worker thread 414 performs whatever cleanup is necessary to prepare the resource 408 for use by another thread (see column 10, lines 61-67). It is noted that the worker thread 414 of Clark uses the resource 408 to perform a task at step 702 while the resource lock is still set. That is, no other worker thread can use the resource while the resource lock is acquired (and not yet released until step 710).

On the other hand, according to some embodiments of the present invention, for example, as illustrated for example in FIG 6A of the present application and described at pages 18-19 of the present application, each worker thread determines at block 610 whether a lock is available. If the lock is available the worker thread acquires the lock at block 612, increments a count at block 614, and then releases the lock at block 616 after the count has been incremented. Once the lock has been released, multiple threads may be allowed to process work concurrently. If there is work to be processed, the worker thread processes the work until there is no work to be processed at block 618. In some embodiments of the present invention the update thread cannot change the state of the shared resource as long as any one of the worker threads is busy processing work (see, for example FIG 6B and pages 19-20 of the present application).

The Clark patent does not disclose at least features of the present invention as claimed of multiple processors arranged to access a shared resource, a single update thread updating or changing the state of the shared resource without requiring serialization of all threads such that an update or change of the state of the shared resource may be made by the single update thread only when none of the multiple worker threads are processing work on the shared resource, and the multiple worker threads able to perform work on the shared resource concurrently.

The Oliver patent discloses a read/write lock management system based upon mutex and semaphore availability. The read/write lock permits a plurality of reader threads to wait simultaneously for access to protected data while only allowing a single writer thread access to a protected data location. Oliver solves a data validity problem relating to a read out-of-schedule during a write from the same data location. As described in relation to the flowchart of FIG 1, the thread A, B, C, D, E example illustrated in col. 4 and described in cols. 4 and 5, and as illustrated in FIGs 3A and 3B and described in cols. 5 and 6, Oliver does not disclose or suggest that multiple read threads concurrently read data. Oliver merely discloses that multiple reader threads may consecutively obtain the mutex at step 114 (not concurrently). For example, one read thread (for example, thread A described in col. 4) reads the protected data at step 124 of FIG

1 and then waits at step 128. This occurs, for example, while thread D is waiting at step 220 of FIG 2 and the other reader threads B, C, E are waiting at step 112 (see, for example, col. 4, lines 49-51). At that point, since thread C has the highest priority it obtains the mutex and reads the protected data. The read threads do not concurrently read the protected data. This fact is also illustrated in reference to FIGs. 3A and 3B and the associated description therein. For example, as illustrated in FIG 3B, only one of the threads (HR, MP, LR) is active at one point in time to read the protected data (as shown by the "active thread" line in FIG. 3B) while the other threads are not active at that same time and do not read the protected data (as shown by the "suspended thread" dotted line in FIG. 3B).

Neither of the Clark patent and the Oliver patent disclose at least features of the present invention as claimed of multiple processors arranged to access a shared resource, a single update thread updating or changing the state of the shared resource without requiring serialization of all threads such that an update or change of the state of the shared resource may be made by the single update thread only when none of the multiple worker threads are processing work on the shared resource, and the multiple worker threads able to perform work on the shared resource concurrently. The Oliver patent merely shows the ability to wait for the shared resource concurrently, not to perform work on the shared resource concurrently. Therefore, withdrawal of the prior art rejections is respectfully requested for at least the reasons set forth above.

In view of the foregoing, the application is considered to be in condition for allowance. Early notification of the same is earnestly solicited. If there are any questions regarding the present application, the Examiner is invited to contact the undersigned attorney at the telephone 815-885-2389.

Respectfully submitted,

[illegible]

March 13, 2006
Date

Robert D. Anderson
Reg. No. 33,826

Intel Americas, Inc.